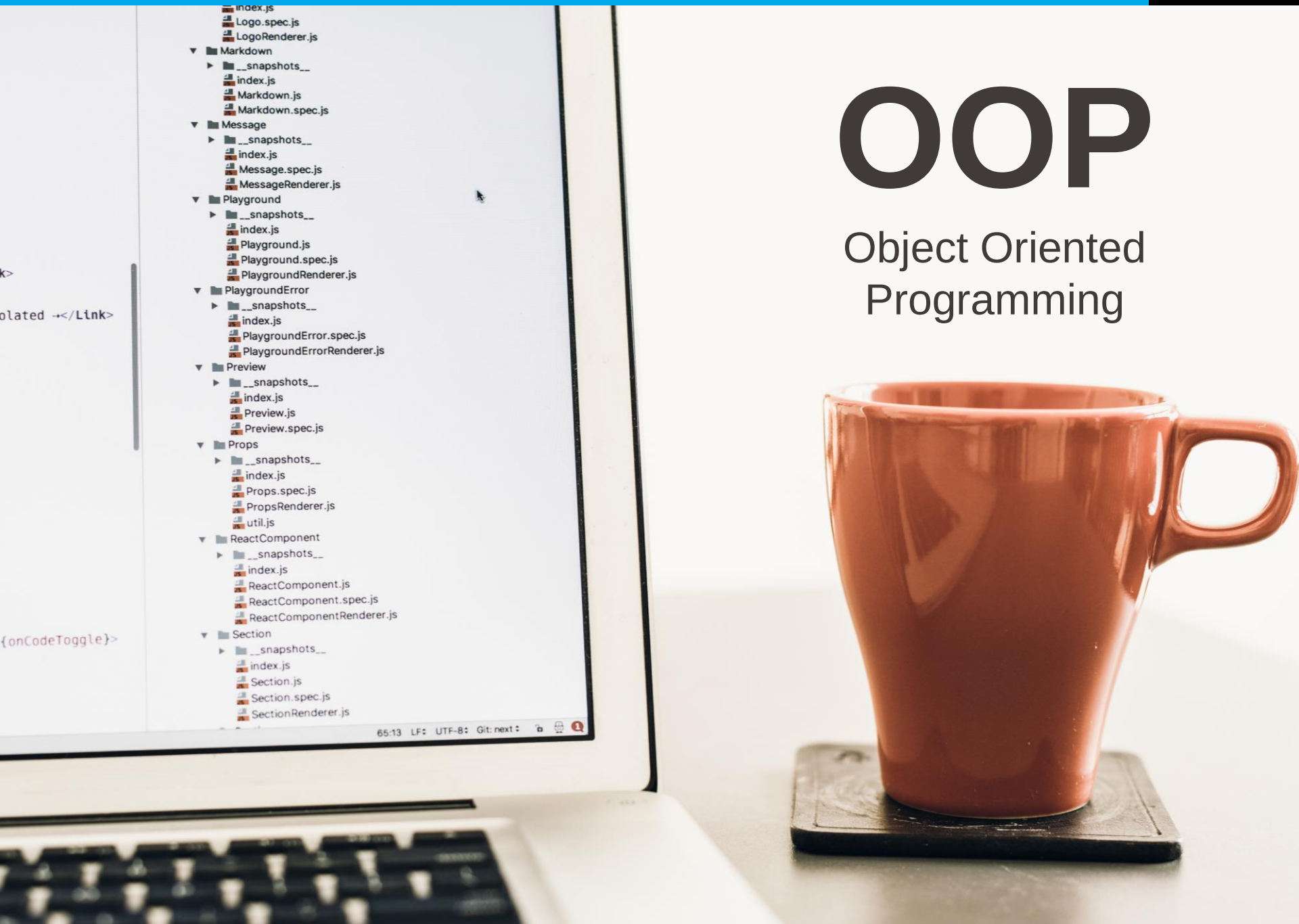
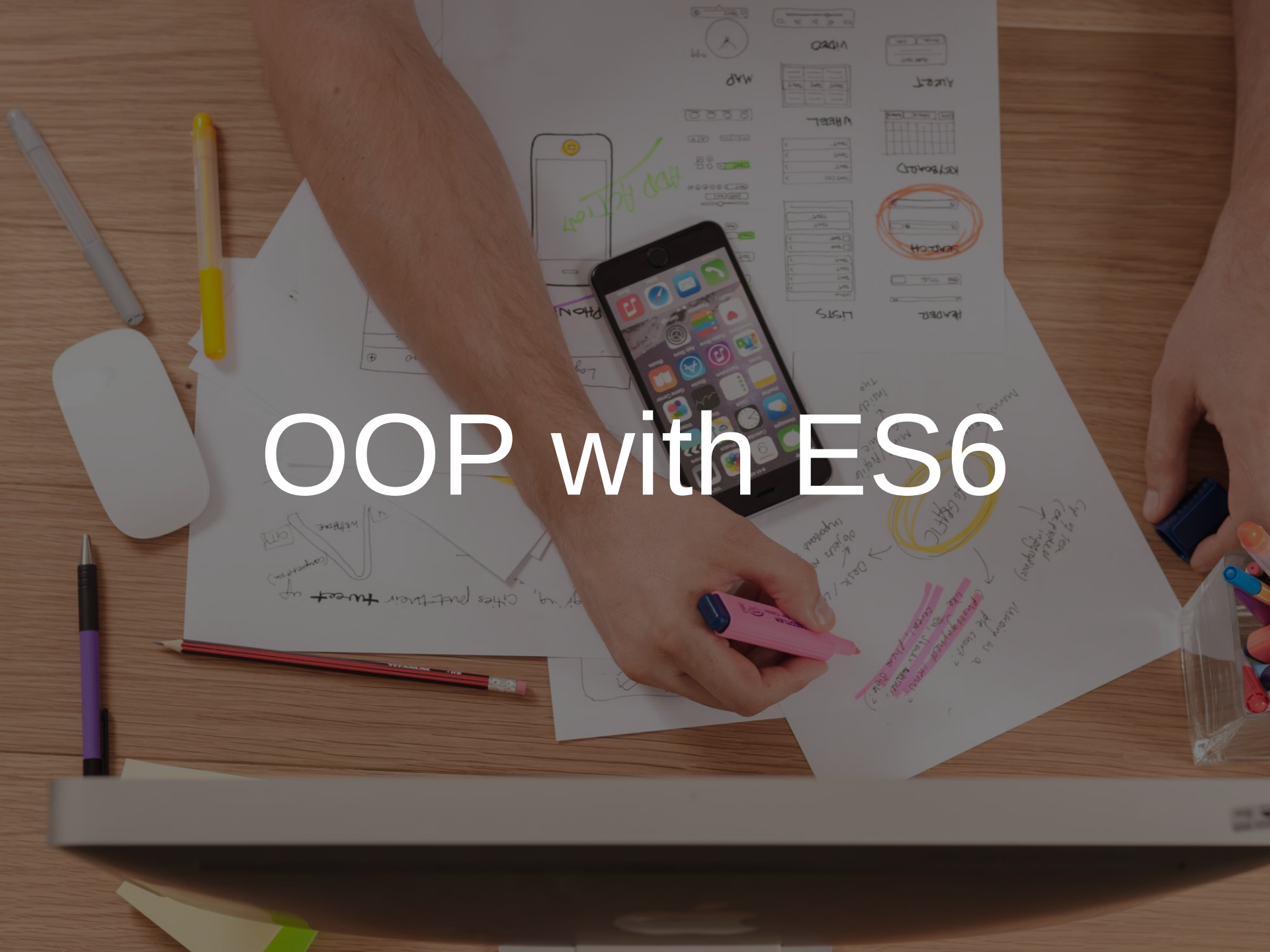
OOP

Object Oriented Programming





OOP with ES6

OOP

Modeling abstract ideas into known domains

For example – a car is an abstract concept.

OOP helps us translate it into the world we know:

- Objects
- Primitive types
- **Function**

Classes and Constructors

ES6 class

We use the **class** keyword to declare a class

Class name starts with a capital letter

```
class Car {
    constructor(color, brand){
        this.color = color;
        this.brand = brand;
    }
}
```

We declare a constructor with a function named constructor.

In the constructor we define only class attributes.

The class is surrounded with curly braces

Class

What is a class?

1. A model for creating objects, with content that describes features and behavior.

2. **Class Members**: Any content of the model, of the class, is called a member of the class.

3. Field members (the class' data) are called **Attributes**

4. Function members are called **Methods**

Car Class

Members of the class

The members of the Car Class:

Attributes:

- Color
- Brand
- Speed

Methods:

- Drive
- Stop

Attributes are also called

variables
properties
data

Car
Color: black
Brand: Ford
Speed: 110
drive(), stop()

Car Class

Members of the class

Car
Color: black
Brand: Ford
Speed: 110
drive(), stop()

The **members** of the Car Class:

Attributes:

- Color
- Brand
- Speed

Methods:

- Drive
- Stop

```
class Car {
   constructor(color, brand){
     this.color = color;
     this.brand = brand;
     this.speed = 0;
   }
   drive(){
     console.log("driving");
   }
   stop(){
     console.log("stopped driving");
   }
}
```

Class Examples

- Each class has its own properties and functions
- We can create any type of class

Table Lamp
isOn
on(), off()

Customer
name, email, address
show details(), update()

Book
Author, title, genre
sell(), restock(), review()

Computer
memory, hard disk
format(), install()


Questions?

classes

OOP with ES6

```
class Animal {
  constructor(name, type){
    this.name = name;
    this.type = type;
  }

  sleep(){
    console.log("I fell asleep");
  }
}
```

We define methods in the class scope

Class Structure

```
class ClassName {
  constructor(param1, param2){
      ...
  }

  method1(){
    ...
  }

  method2(param1){
    ...
  }

}
```

The built-in constructor method is first!

Then the rest of the functions. The order is not important.

classes

1 Big No-No!

```
class ClassName {
  constructor(param1, param2){
      ...
  }

  if (this.name.length > 0){
    console.log("No no no!");
  }

  method1(){
    console.log("here is ok");
  }

}
```

Only in the constructor

You can **NOT** write js code outside of a class method!!

Or in any other class method

Methods

Methods are functions associated to a class

We can add many methods to a class

Any instance of the class can invoke any function declared in the class:

```
class Animal {
  constructor(name, type) {
    this.name = name;
    this.type = type;
  }

  sleep() {
    console.log("I fell asleep");
  }
}

let myDog1 = new Animal("Bella", "dog");
myDog1.sleep();//I fell asleep
```

Methods

We had:

```
sleep(){
  console.log("I fell asleep");
}
```

But what if we wanted to type the name of the animal too
Instead of “I fell asleep” => “Bella fell asleep”?
When using objects literals we can do it like **this**:

```
let bella = {
  name: "bella",
  type: "dog",
  sleep: function(){
    console.log(this.name + " fell asleep");
  }
}
bella.sleep();
```

Note that this will work as expected only when calling the function with the context of the object `bella`

The ‘this’ keyword with the `class`

this is used to refer to the instance itself

- **Inside a constructor** – it refers to the instance being created.
- **Inside a method** – it refers to the instance already created.

```
class Animal {
  constructor(name, type) {
    this.name = name;
    this.type = type;
  }

  sleep() {
    console.log(this.name + " fell asleep");
  }
}

let bella = new Animal("bella", "dog");
bella.sleep(); // bella fell asleep
```

A method can get parameters

Methods can be with or without parameters. (like any js function)

```
sleep(hours) {
   //checking if we got a numeric parameter
   if (typeof hours === "number") {
      console.log(this.name + "fell asleep for" + hours + "hours");
   } else {
      console.log(this.name + " fell asleep");
   }
};
```

```
myDog1.sleep();
```

```
bella fell asleep
```

```
myDog1.sleep(8);
```

```
bella fell asleep for 8 hours
```

Exercise – Player did win

Create Constructor and a Method

Your task: create Player class.

It should have the following attributes:

name – will be an argument of the constructor

score – set to 0 at creation
and a method didWin - that checks if the score is higher than 30.

1. Create a player instance.
2. Set its score to 32.

3. Invoke the didWin method.

Methods

A method is just another property of the object

```
class Animal {
  constructor(name, type) {
    this.name = name;
  }

  sleep() {
    console.log("I fell asleep");
  }
}

let myDog1 = new Animal("Bella");
myDog1.sleep();//I fell asleep
```

We can access it like any other property

Methods

A Method Can Invoke Another Method

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  sleep() {
    console.log(this.name + " fell asleep");
  }
  eat(){
    console.log(this.name + " is eating");
  }
  eatAndSleep(){
    this.eat();
    this.sleep();
  }
}

let myDog1 = new Animal("Bella");
myDog1.eatAndSleep();
```

Here we are inside `eatAndSleep` method and we invoke other class methods: `eat` and `sleep`.

```
> myDog1.eatAndSleep()
  Bella is eating
  Bella fell asleep
```

Exercise – Check if player won

Method Invoking a Method

Your task: use the Player class.

Add a `checkIfPlayerWon` method that checks if a player won and if he did prints a message

"player <player name> won!"

player name – the name of the player.

Then:

1. Create a player instance.
2. Set its score to 32.
3. Call the `checkIfPlayerWon` method

Object Review

- An object is a collection of keys and values.
- Objects' properties can have every type

```
let student = {
  name: "John Doe",          ← string
  age: 16,                   ← number
  grades: [97,78,82],        ← array
  school: {                  ← Object
    name: "Beverly Hills",
    address: "241 S Moreno Dr, Beverly Hills, CA 90212, USA"
  },
  getHeighestGrade: function(grades){   ← function
    let highGrade = 0;
    grades.forEach(grade => {if (grade > highGrade){highGrade = grade}});
    return highGrade;
  },
};
```

A Student Class

```
class Student {
  constructor(name, age, school) {
    this.name = name;
    this.age = age;
    this.grades = [];
    this.school = school;
  }

  getHeighestGrade() {
    let highGrade = 0;
    this.grades.forEach(grade => {
         if (grade > highGrade)    {
           highGrade = grade
         }
       });
    return highGrade;
  }
}
```

School is an object

Create a Student

```
let school = {
  name: "Beverly Hills",
  address: "241 S Moreno Dr, Beverly Hills, CA 90212, USA"
};
let student = new Student("John Doe", 16, school);
```

Or passing the school object on the fly:

```
let student = new Student(
  "John Doe",
  16,
  {
      name: "Beverly Hills",
      address: "241 S Moreno Dr, Beverly Hills, CA 90212, USA"
  }
);
```

Create a Student… now with the ES6 class

```
class School {
  constructor(name, address) {
    this.name = name;
    this.address = address;
  }
}


let beSchool = new School("Beverly Hills", "241 S Moreno Dr,
Beverly Hills, CA 90212, USA");



let student = new Student("John Doe", 16, beSchool);
```

```
> student.school
<· ▶ School {name: "Beverly Hills", address: "241 S Moreno Dr, Beverly Hills, CA 90212, USA"}
```

Object parameter

We can even pass object of the same type as a parameter:

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  makeFriend(animal) {
    console.log(this.name+" and " + animal.name + " are friends");
  }
}

let blacky = new Animal("blacky"); //create an animal instance
let bella = new Animal("bella"); //create another animal instance
blacky.makeFriend(bella); //blacky and bella are friends
```

We pass an argument of type Animal.
animal is an instance of Animal

Notice the difference?

myDog1.sleep

```
> myDog1.sleep
<· ƒ (){
      console.log("I fell asleep");    }
```

Reading a property

myDog1.sleep()

```
> myDog1.sleep();
  I fell asleep
```

Calling a method


Questions?

OOP Cheat Sheet

Define a Class

```
class Animal {
  constructor(name, type){
    this.name = name;
    this.type = type;
  }

  greet(){
    console.log(this.name + " says hi");
  }

}
```

Referencing the instance with “this” keyword

Create an instance

```
let blacky = new Animal("blacky", "dog");
```

Get property

```
blacky.type;
```

Update property

```
blacky.type = "cow";
```

Invoke instance method

```
blacky.greet();
```

Iterating through properties

```
for(let prop in bella){
  console.log("property: " + prop + ", value: " + bella[prop]);
}
```